UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

20|21

Trabajo fin de grado

Monitorización de un conjunto de eventos producidos en sistemas

Celia Montarelo Mardomingo

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

excelencia Campus Internacional UAM CSIC

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Doble Grado en Ingeniería Informática y Matemáticas

TRABAJO FIN DE GRADO

Monitorización de un conjunto de eventos producidos en sistemas

Autor: Celia Montarelo Mardomingo
Tutor: Javier Aracil Rico
Ponente: Javier Aracil Rico

octubre 2020

Celia Montarelo Mardomingo
Monitorización de un conjunto de eventos producidos en sistemas

Celia Montarelo Mardomingo
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mi familia y amigos

Una máquina puede hacer el trabajo de 50 hombres corrientes. Pero no existe ninguna máquina que pueda hacer el trabajo de un hombre extraordinario.

Elbert Hubbard

Agradecimientos

Antes de entrar en materia me gustaría agradecer, en primer lugar, a Naudit HPCN por, desde hace un año, confiar en mí para poder llevar a cabo mis prácticas curriculares, las cuales culminaron con la oportunidad de poder desarrollar los trabajos de fin de grado, tanto de informática como de matemáticas. Javier Aracil, gracias por todo. También mi agradecimiento a mis compañeros en este proyecto: Diego Sainz y Tomás Hernández, sin su labor este trabajo no sería lo mismo.

Del mismo modo me gustaría agradecer a mi familia y amigos su apoyo, paciencia y confianza, y más en esta situación actual tan anómala. Ellos me han dado ánimo y fuerza para realizar este proyecto, especialmente en los momentos más estresantes.

Resumen

Actualmente la mensajería, en especial la demanda de paquetería, ha sufrido un aumento descomunal debido, entre otras cosas, a la aparición de un gran número de empresas de compra-venta *online*. Es más, este año, y debido a la situación insólita a la que nos enfrentamos, el aumento de los servicios *online* ha sido enorme, en comparación con los años anteriores.

La problemática recae en la insuficiencia del procedimiento de monitorización de las distintas transacciones o eventos. En nuestro caso concreto, una gran compañía logística española analizaba los sistemas que transmiten estos eventos de forma independiente, lo cual provocaba, entre otras cosas, la pérdida o retraso de los paquetes sin que la propia empresa tuviera conocimiento de estas incidencias causando así el descontento de los clientes.

Nuestra propuesta va en esa línea para resolver esta deficiencia y se fundamenta en unificar esta monitorización del ciclo de vida de los eventos transmitidos entre los sistemas, teniendo como finalidad facilitar el trabajo a los analistas de la empresa logística. Incluso, gracias a las alertas configuradas que hemos creado, los analistas, podrán identificar los retrasos o pérdidas producidos en los envíos y así, poder realizar la notificación a los clientes a tiempo.

En definitiva, este proyecto proporciona una herramienta que reduce el trabajo de los analistas, ya que, toda la información sujeta a análisis estará accesible desde la interfaz web, caracterizada por su uso intuitivo. Luego nuestro proyecto cumple con el objetivo principal (unificar la monitorización y detectar incidencias) y lo lleva a la práctica.

Palabras clave

Interfaz, monitorización, evento, ventana emergente, nodo, alarmas y compañía logística

Abstract

Currently the courier service, especially the demand for shipping packages, has suffered a huge increase due, among other things, to the great appearance of e-commerce companies. Moreover, this year, due to the unusual situation we are facing, the increase in online services has been massive compared to previous years.

The problem has been the insufficiency of the monitoring procedure for the different transactions or events. Specifically, a large Spanish logistics company analyzed the systems that transmit these events independently, which caused, among other things, the loss or delay of packages without the company itself being aware of these incidents, what caused the dissatisfaction of the company's clients.

Our proposal to solve this deficiency is based on unifying this monitoring of the life cycle of the events transmitted between systems, with the aim of facilitating the work of the logistics company analysts. Even thanks to the configured alerts that we have created, the analysts will be able to identify delays or losses in shipments and thus be able to notify to the clients on time.

In conclusion, this project provides a tool that reduces the work of the analysts, since all the information subject to analysis will be accessible from the web interface, characterized by its intuitive use. Then our proyect meets the main objective (to unify monitoring and detect incidents) and puts its into practice.

Keywords

Interface, monitoring, event, pop-up, node, alarms and logistics company

Índice

Listas

Lista de figuras

1

Introducción

Este proyecto presenta un sistema de monitorización de la trazabilidad de un conjunto de datos, entre distintos sistemas, con el fin de proporcionar mayor información acerca del tráfico de datos.

Actualmente, la aplicación propuesta está en uso en un conjunto de sistemas y se pretende extender su utilización a un conjunto mayor, con el fin de unificar la monitorización de los flujos de datos, en una sola aplicación.

1.1. Motivación

En los últimos años, se ha producido un aumento en el uso de los servicios de mensajería y paquetería, debido a la aparición de empresas de comercio electrónico como Amazon, AliExpress, etc., y a la mayor accesibilidad a dispositivos electrónicos como Smartphones, portátiles, tabletas, etc., lo que ha producido que tanto las empresas de servicio postal y paquetería, como las compañías logísticas, hayan aumentado sus transacciones.

En el caso de la empresa logística española que solicitó nuestros servicios, esta mensajería es distribuida entre distintas aplicaciones y sistemas, facilitando así la comunicación con agentes externos. Sin embargo, la monitorización de los datos es insuficiente puesto que monitoriza sólo sistemas, lo cual ha generado varios problemas en la monitorización, como es la incapacidad de detectar la pérdida de información entre sistemas internos, la inexistencia de alertas ante el tiempo excesivo o la disminución en el tráfico de datos; las incidencias, en algunos casos, son detectadas antes por los clientes que por la propia compañía logística.

Este trabajo presenta un sistema de monitorización de la traza de los datos de envíos en los sistemas de dicha empresa logística española, con el fin permitir visibilizar el ciclo de vida de la información de los envíos realizados.

La aplicación presentada pretende resolver un problema real, ofrece un sistema de monitorización de la traza de los datos de los envíos, en los sistemas de información de la compañía logística, con el fin de facilitar el trabajo de los analistas del tráfico de red. Estos se encargan de prevenir los errores

producidos en la transmisión de los datos, para que el usuario final, cliente que realiza la compra-venta de un artículo, no se vea afectado por problemas asociados a la red.

1.2. Objetivos

El objetivo de este proyecto es desarrollar un sistema de monitorización de la traza de los distintos eventos generados en los sistemas de una compañía logística española, en particular, se monitorizarán los procesos/eventos de tipo: Pre-registro, admisión, entregado y devuelto, con el fin de controlar el ciclo de vida de la información.

Los principales objetivos de sistema de monitorización son:

- Detectar pérdidas de información mediante la visión global de los procesos en función de la información de los *logs* de los distintos sistemas.
- Detección de la disminución o de un gran aumento en el tráfico de la mensajería.
- Detección temprana del aumento en el tiempo de transito entre los distintos sistemas que pueda indicar la aparición de un cuello de botella.
- Asegurar la entrega de los envíos en los umbrales de tiempo establecidos.

El proyecto se desarrollará de forma incremental, mostrando varios prototipos de la aplicación desarrollada para disponer así de un *feedback* de los clientes.

Con respecto al diseño, se desarrollará un sistema modular, para facilitar el mantenimiento, que sea escalable, de alto rendimiento y eficiente en el consumo de recursos. Además tendrá una *interfaz responsive* adecuada para los distintos dispositivos utilizados en la monitorización.

1.3. Estructura de la memoria

La memoria ha comenzado con una introducción al proyecto donde hemos hablado de la motivación y los objetivos del mismo.

En el siguiente capítulo explicaremos las tecnologías empleadas para la monitorización de la traza de los eventos, utilizadas para la realización de este proyecto.

Continuaremos explicando los requisitos del proyecto y la arquitectura seguida en el Capítulo 3.

En el Capítulo 4 explicaremos el diseño seguido para realizar la aplicación y las conexiones con la base de datos. También hablaremos de los archivos PHP involucrados en el proceso de autenticación de la aplicación y en los procesos de configuración de las alarmas y del diagrama de flujos.

Seguiremos con el Capítulo 5 donde explicaremos las pruebas realizadas en la aplicación y mostraremos los resultados de las mismas.

Finalizaremos el trabajo con las conclusiones obtenidas tras el desarrollo de la aplicación.

2

Estado del arte

Este proyecto presenta un nuevo sistema de monitorización de la traza de los eventos que se transmiten entre los distintos sistemas de una compañía logística española. Para ello se han combinado distintas herramientas.

2.1. Tecnologías empleadas

La aplicación deberá soportar una gran ingesta de datos, para ello se ha utilizado el motor de búsqueda y análisis *Elasticsearch*, que permite la ingesta masiva de datos y ofrece un acceso instantáneo a los datos.

La información de los eventos a monitorizar se introducirá en *Elasticsearch*. Una vez que los datos del evento se han introducido en *Elasticsearch*, aparecerán en las gráficas generadas en la aplicación, una de las ventajas de *Elasticsearch* es la rapidez con la que realiza las búsquedas, permitiendo así que la interfaz consulte esta información y muestre el resultado de la consulta de forma casi instantánea.

Para visualizar los datos introducidos en *Elasticsearch* se ha utilizado la herramienta *Kibana*, aplicación que facilita la búsqueda, análisis y visualización de los datos indexados en *Elasticsearch*, que permite la creación de *dashboards* personalizados que contengan gráficos y tablas con la información actualizada de los eventos.

Se ha utilizado el sistema *PostgreSQL* para mantener los tiempos de los eventos hasta que finalizan. Este sistema de base de datos relacional de objetos es concurrente, permitiendo así la escritura y lectura de una misma tabla de datos y proporcionando así integridad transaccional de los datos.

Para desarrollar la aplicación se ha utilizado el *framework* de código abierto Angular, en particular Angular 9, debido a las facilidades que ofrece para crear aplicaciones de una sola página (una misma página interacciona con el cliente y reescribe esa misma página, dinámicamente, con los nuevos datos). Debido a esto las aplicaciones en Angular, tras cargar la página principal, tienen un tiempo de respuesta prácticamente instantáneo lo que permite una navegación más fluida.

Angular se caracteriza por la combinación del código *HTML* y *CSS* con el lenguaje de código abierto *TypeScript* (basado en *JavaScript*), con ello Angular separa las partes *front-end* y *back-end* de la aplicación. Angular también se caracteriza por presentar una aplicación modular (en función de los componentes) y escalable (en general los componentes son independientes unos de otros y un cambio o actualización en uno de ellos no afecta al conjunto).

Dentro de las librerías utilizadas, destacamos el uso de dos de ellas, la librería de visualización de gráficos de código abierto *ngx-graph*. Esta librería permite la creación de diagramas de nodos, además se pueden definir nuevas estructuras de datos, permitiendo así personalizar los nodos y enlaces que aparecen en el diagrama según las exigencias del cliente.

Para generar el diagrama de nodos, será necesario tener una lista con los nodos a crear, esta lista se ha almacenado en un fichero *JSON* (JavaScript Object Notation) en los servidores. La aplicación accederá a la lectura y escritura de este documento mediante un servidor *PHP*, para ello Angular proporciona una *API* cliente *HTTP* para realizar llamadas *get/post* para realizar *API REST*, desde las distintas componentes de la aplicación, para poder cargar/modificar los datos almacenados en el servidor.

Partiendo de esta librería se han definido los nuevos tipos de nodos y sus características en función del fichero *JSON*, para su posterior carga en el diagrama de la aplicación.

Otra de las librerías utilizadas de Angular es *ngx-chart*, esta librería permite la creación de gráficos dinámicos, con ello se podrán generar gráficos con la información actualizada de *Elasticsearch*. Además esta librería se caracteriza por sus estilos personalizables mediante *CSS* que dan a los gráficos una mejor apariencia.

3

Análisis

En esta sección describiremos los requisitos funcionales y no funcionales que debe seguir la aplicación, estos se han definido tras una serie de reuniones iniciales con el cliente.

También explicaremos la arquitectura que se va a seguir para el desarrollo del sistema.

Previamente vamos a indicar los usuarios de la aplicación que serán de tres tipos según su rol:

- **Administrador**: Será el encargado de crear y configurar las alarmas de tipo negocio y administración, además de editar el diagrama de flujos. Es el usuario que posee los permisos para utilizar toda la funcionalidad de la aplicación.
- **Consultor**: Usuario que podrá consultar las alarmas creadas (podrá acceder al gestor de alarmas pero no podrá editarlas). Se encargará de analizar el flujo de los eventos desde el diagrama y de responder a las alarmas mostradas desde el diagrama.
- **Visualizador**: Usuario que solo podrá acceder al diagrama y se encargará de identificar y responder a las alarmas producidas sobre este.

3.1. Requisitos

En este apartado indicaremos la lista de requisitos de la aplicación, comenzaremos por los requisitos funcionales y posteriormente pasaremos a los requisitos no funcionales.

3.1.1. Requisitos funcionales

RF-1.– Asegurar la transmisión de la información de cada uno de los procesos (también llamados eventos), durante su ciclo de vida, esta transmisión se podrá verificar desde el diagrama que representa los flujos a monitorizar, la Figura 4.2 muestra este diagrama. Cada flujo tendrá una serie de nodos a través de los cuales se transmite la información de los procesos.

RF-2.– La aplicación deberá mostrar una pantalla simple donde se apreciarán los flujos a monitorizar y la existencia de anomalías sobre ellos. Se podrá acceder a información más detallada pulsando en los objetos del diagrama.

RF-3.– Se podrá configurar una serie de alertas en función de la transmisión de los procesos a lo largo del flujo. Dichas alertas, parametrizables según el tipo de usuario, se realizarán entre pares de sistemas.

Ante la aparición de anomalías se destacará, en el diagrama, el flujo donde se han producido. Se seguirá un formato a la hora de resaltar los nodos:

- **Rojo**: Se ha superado el umbral de la alerta, el nodo resaltará en color rojo.
- **Amarillo**: Casi se supera el umbral de la alerta, el nodo resaltará en color amarillo.
- **Verde**: No se aprecia ninguna anomalía, los nodos tendrán su apariencia normal.
- **Negro**: No hay conexión, el nodo resaltará en color negro.

RF-4.– En caso de producirse una anomalía, se mostrará un icono en el diagrama, siguiendo el código indicado en **RF-3.**, que irá cambiando o desaparecerá en función de la alerta producida. Al pulsar sobre los iconos, se deberá mostrar el flujo sobre el que se ha producido la alerta.

RF-5.– El gestor de alarmas tendrá dos versiones, el gestor de alarmas de administrador (*Alert Manager Administrador*) y el gestor de alarmas de negocio (*Alert Manager Negocio*).

El usuario de tipo **administrador** podrá configurar las alarmas de negocio y administrador.

El usuario de tipo **consultor** podrá acceder sólo al gestor de alarmas de negocio y podrá ver las alarmas configuradas.

El usuario de tipo **visualizador** no podrá acceder a los gestores de alarmas, pero si visualizará las alarmas producidas en el diagrama.

El administrador podrá establecer umbrales de alarmas más sensibles desde el gestor administrador, estas alarmas aparecerán marcadas en el diagrama en los usuarios de tipo administrador. A su vez, el administrador podrá configurar alarmas con umbrales menos sensibles desde el gestor de negocio, estas alarmas, en caso de producirse, aparecerán en el diagrama de los usuarios consultor y visualizador.

RF-6.– Se generarán alarmas si algún *log* está mal formado o si dejan de escribirse en el servidor.

RF-7.– La aplicación deberá mostrar una *interfaz responsive* que se adapte a los distintos dispositivos utilizados para monitorizar. La navegación entre las distintas pantallas y el diagrama ha de ser de usabilidad sencilla e intuitiva, pudiendo acceder a distintos niveles de detalle al pulsar en los objetos del diagrama.

RF-8.– Los eventos a monitorizar tendrán un estado asociado, existen tres tipos de estados:

- **Pre-registro**: Representa el estado de eventos que han entrado en el sistema, pero aún no han sido admitidos.
- **Admisión**: Estado de los eventos que han pasado el sistema de admisión y aún no se han entregado.
- **Entregado y/o devuelto**: Son los estados finales de los eventos que han llegado al destino y aquellos que han sido devueltos.

RF-9.– Los flujos de eventos se deberán controlar por pares de sistemas (sistema origen y sistema destino).
Al pulsar sobre estos, se deberá mostrar en un cuadro de diálogo que muestre una gráfica con el número de procesos que entran desde el flujo de entrada y los que salen por el flujo de salida, según el tipo de proceso (pre-registro, admisión, entregado o devuelto).

RF-10.– Se deberá poder buscar los mensajes con la información de los procesos a monitorizar mediante su código de envío, con posibilidad de descargar los *logs* asignados a ese código.

RF-11.– La BBDD que almacenará los registros de trazabilidad de los procesos deberá ser accesible desde otros proyectos que requieran esa información, por lo que esta BBDD requiere alta disponibilidad.

RF-12.– Los distintos sistemas implicados en la monitorización deberán ser capaces de compartir los datos entre ellos.

RF-13.– Una vez terminada la revisión del la aplicación, se designará una persona como soporte de la misma durante un mes, encargado de atender las peticiones acerca de la documentación o el desarrollo de la aplicación.

RF-14.– El sistema debe ser capaz de mostrar, copiar y procesar un volumen aproximado de unos veinte millones de mensajes al día.

RF-15.– La aplicación permitirá a los usuarios de tipo administrador editar el diagrama desde una ventana emergente de configuración, donde podrán añadir, editar o eliminar un nodo del diagrama. Además se proporciona la opción de modificar la posición de los nodos del diagrama.

3.1.2. Requisitos no funcionales

RNF-1.– Seguridad: La aplicación permitirá el acceso sólo a los usuarios autorizados. Además se restringirá, en función del rol del usuario, el acceso a parte de la funcionalidad de la aplicación.

RNF-2.– Usabilidad: La interfaz de la aplicación mostrará un diagrama sencillo desde el cual podremos acceder de manera intuitiva al resto de funcionalidad de la aplicación, facilitando así su uso a los usuarios.

RNF-3.– Rendimiento: La aplicación presenta un tiempo de respuesta inferior a 5 segundos ante la carga, búsqueda y actualización de datos, independientemente del número de usuarios que accedan a la misma.

RNF-4.– Mantenimiento: La aplicación se ha desarrollado de manera modular, lo cual permite facilitar las actualizaciones a versiones más modernas.

RNF-5.– Fiabilidad: El sistema realizará periódicamente el almacenamiento y el copiado de los *logs* con la información de los eventos en los servidores, en caso de que estas copias no se realicen correctamente, se generará una alarma. Además se mantendrán varias copias de seguridad, del diagrama de flujos y de las alarmas configuradas por los usuarios, en los servidores por si se produce un error o se desea deshacer un cambio.

RNF-6.– Portabilidad: La aplicación podrá adaptarse sin pérdida de calidad a los distintos dispositivos utilizados para la monitorización, entre ellos *Tablet*, *Smartphone*, *PC*, *PDA*, etc.

3.2. Arquitectura

Las Figuras 3.1 y 3.2 representan esquemáticamente la arquitectura del sistema.

Los datos a monitorizar entran al sistema desde los servidores de la empresa logística española, estos se introducen mediante una pieza de software *ad-hoc* (código desarrollado en *C* y en *AWK* para este cliente) que permite copiar, de forma eficiente, ficheros muy grandes.

Se realizará un primer procesado sobre estos datos masivos para extraer la información relevante. Tras este proceso, se importará la información importante de los eventos a *Elasticsearch* (motor de ingesta y búsqueda de todo tipo de datos a gran velocidad).

También se irán guardando y actualizando las marcas de tiempo asociadas a los eventos, es decir, el tiempo que lleva el evento dentro del sistema. Estos tiempos se guardarán en *PostgreSQL* hasta que el evento finalice.

Una vez que el evento ha finalizado, se importa a *Elasticsearch* y se borra de la base de datos relacional (*PostgreSQL*). Existirán dos cluster (agrupaciones) independientes en *Elasticsearch*, uno de ellos se encargará de almacenar los datos procesados mientras que el otro será el encargado de almacenar los *logs* (datos con información de los eventos).

Un *Cron* (administrador de procesos en segundo plano, *daemon*) analizará la base de datos re-

lacional cada cierto tiempo, el cual dependerá del volumen de los datos, con el fin de identificar los procesos que superen los tiempos de paso configurados y lanzar una alarma.

Mediante el *Cron* también se comprobarán cada cierto tiempo las alarmas establecidas con el gestor de alarmas de la interfaz, para poder actualizar, añadir o desactivar las alarmas existentes almacenadas en *Elasticsearch*.

La interfaz web se encargará de mostrar de forma esquemática el ciclo de vida de los eventos a monitorizar, en la pantalla inicial se mostrará un diagrama con las distintas etapas por las que pasarán los eventos.

La interfaz estará conectada con *Kibana* (para facilitar la creación y actualización de gráficas asociadas a las etapas de los eventos) y *Elasticsearch* para la búsqueda de datos y para configurar y mostrar las alarmas.

Esta arquitectura proporciona un eficiente procesado y copiado de *logs* además de disminuir las pérdidas producidas en el monitoreo de los eventos, esto es gracias a las copias realizadas de los mismos. Además gracias al uso de *Elasticsearch* se pueden realizar consultas rápidas sobre los datos de los eventos y las alarmas producidas. El generador de gráficos *Kibana* permite visualizar los datos de monitoreo indexados en *Elasticsearch* mediante gráficas.

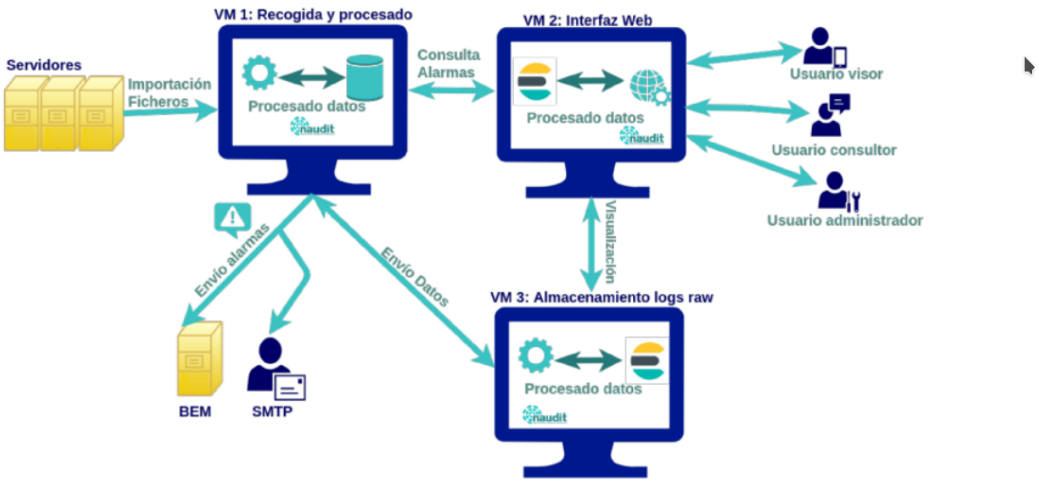


Figura 3.1: Diagrama simple de la arquitectura del sistema de monitorización del ciclo de vida de la información

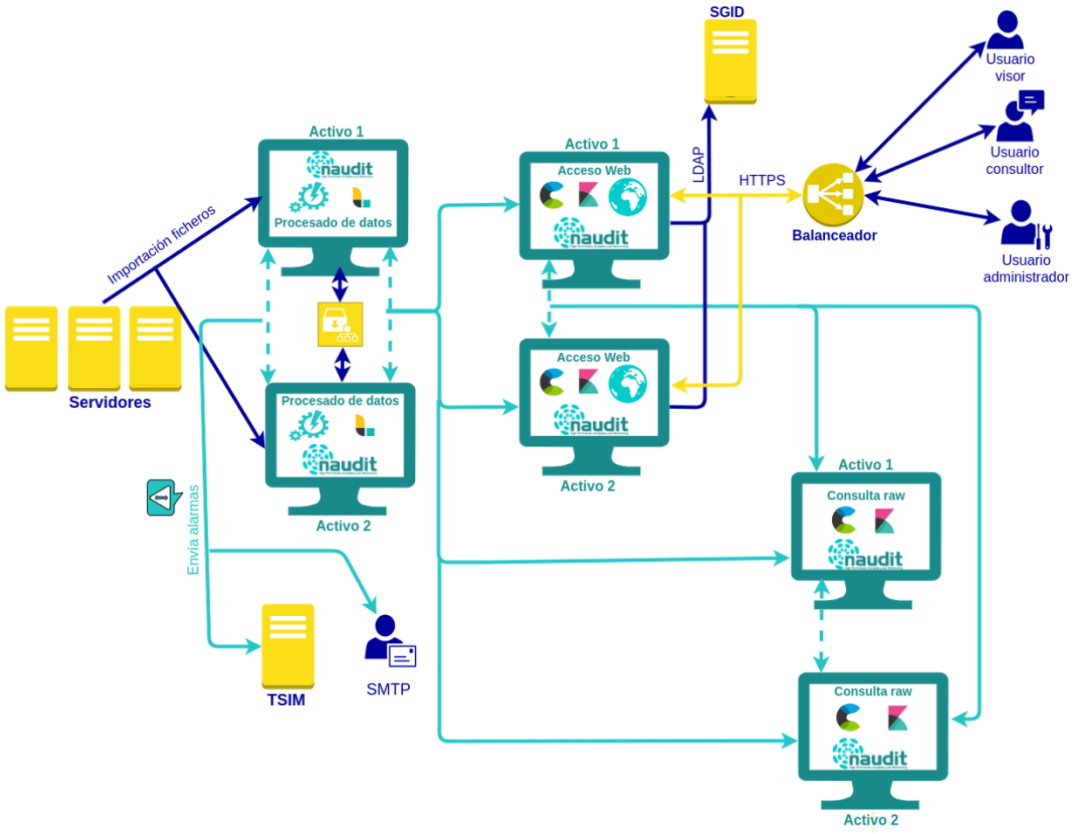


Figura 3.2: Arquitectura del sistema de monitorización del ciclo de vida de la información

4

Desarrollo y diseño

En este apartado describiremos las decisiones de diseño seguidas para realizar este proyecto, junto con los cambios de diseño que se han tenido que realizar ante la aparición de nuevos requisitos, o los cambios sobre los mismos, aportados por los clientes.

En este trabajo nos centraremos en la parte de diseño de interfaz gráfica, es decir, en la parte de *front-end* y *back-end* de la aplicación web, y en las conexiones con la base de datos de *Elasticsearch*, para el monitoreo de los eventos producidos en el sistema.

4.1. Diseño

4.1.1. Angular

Antes de entrar en el diseño de la aplicación, haremos una breve introducción a Angular.

Para desarrollar la interfaz hemos utilizado el *framework* de código abierto Angular, desarrollado en el lenguaje *TypeScript*. Angular se caracteriza por su desarrollo modular basado en componentes y por ser una aplicación de una sola página, desde la cual se puede navegar entre las distintas componentes de la aplicación, sin necesidad de actualizar la página (recargar el navegador).

Una aplicación en Angular contiene un módulo raíz encargado del arranque de la misma. La arquitectura de la aplicación está formada por distintos módulos independientes, formados por componentes que a su vez pueden estar formados por otros componentes.

Los componentes están formados por:

- Las **vistas o templates**, plantillas *HTML* personalizadas que proporcionan la estructura visual de la aplicación.
- La **clase** es una estructura formada por un conjunto de variables o atributos y los métodos que trabajan sobre estos, definiendo así el comportamiento de los componentes. Las instancias de las clases, forman objetos.
- **Metadata** se encarga de asociar la *clase* del componente con la *template* que define la vista, además proporciona una etiqueta *HTML* para referenciar el componente.

Los componentes de la aplicación se comunican con elementos externos mediante los *services*,

que a su vez proporcionan nueva funcionalidad a los componentes en los que se declaran.

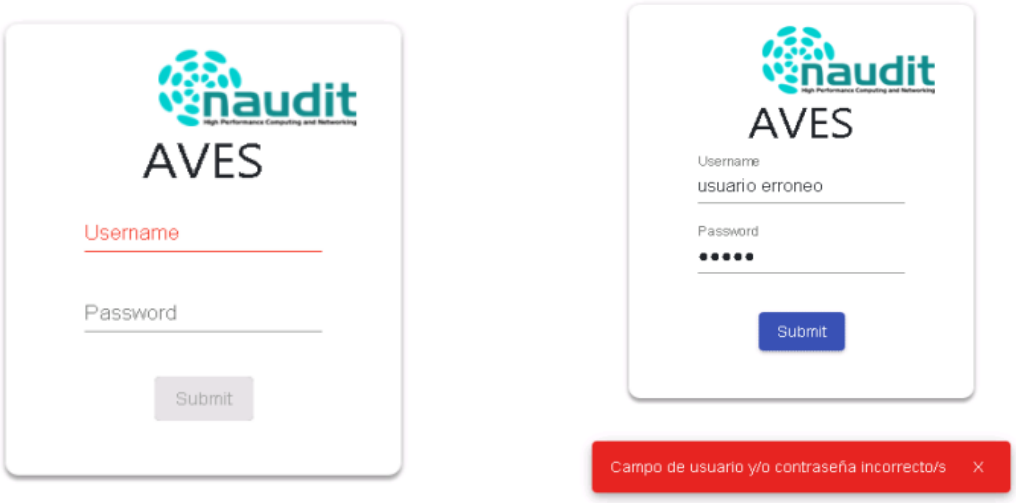
Además mediante el *data binding* se enlazan los elementos de la clase con su *template*, es decir, el *data binding* es la sincronización de datos entre la clase del componente y su vista, permitiendo así que si la clase cambia, la *template* asociada a esa clase cambie también. Esta sincronización tiene un coste de procesamiento bajo.

Por lo tanto, Angular mediante las componentes, los *services* y el *data binding*, permite que el código sea modular, reutilizable, eficiente y de fácil mantenimiento.

4.1.2. Login de la aplicación

Tras acceder a la aplicación desde el navegador, será necesario rellenar el formulario que muestra la Figura 4.1(a), que comprobará si el usuario y la contraseña son correctos. Para ello se realiza una consulta a un archivo *PHP* y se procesa la respuesta del mismo. En el caso de que el usuario y/o la contraseña sean incorrectos, se indicará con un mensaje tal y como muestra la Figura 4.1(b).

Si la autenticación es correcta, se accederá a la pantalla principal de la aplicación, la Figura 4.2 muestra esta pantalla.



(a) El formulario del login.

(b) Respuesta al intentar acceder con usuarios erroneos.

Figura 4.1: Ventana con el formulario a rellenar para acceder a la aplicación.

4.1.3. Diseño del diagrama de monitorización de eventos

La parte principal de la aplicación consiste en el diagrama de la Figura 4.2, por lo tanto el diseño de la aplicación comenzó centrándose en esta parte.

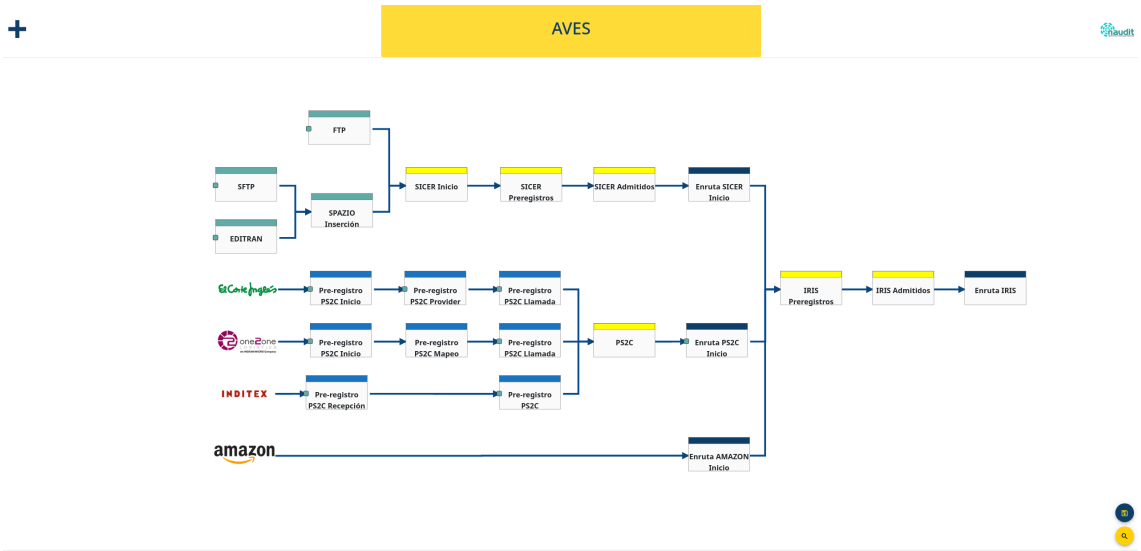


Figura 4.2: Pantalla principal que muestra el diagrama con las etapas de monitorización del ciclo de vida de los eventos.

Para desarrollar el diagrama se ha utilizado la librería *ngx-graph* de Angular, que permite diseñar fácilmente diagramas mediante la definición y modificación de estructuras de datos.

En el diagrama a realizar, se pueden observar distintos elementos:

- Los **nodos**, que se representan en forma de cajas de colores enlazadas. Cada nodo representa una etapa por la que se transmiten los eventos.
  Con el fin de resolver los cambios en los requisitos con respecto a la funcionalidad de los nodos, decidimos distinguir entre cuatro tipos de nodos:
  - Nodos de tipo **Ficheros**, en forma de cajas con la franja superior de color azul-verde. Los nodos de este tipo se caracterizan por ser clickables (en el apartado de ventanas emergentes se explicará esta característica).
  - Nodos de tipo **Enruta**, son las cajas con la franja superior de color azul oscuro. Estos nodos, al igual que los anteriores son clickables.
  - Nodos de tipo **Pre-registros**, son las cajas con la franja superior de color azul claro. Estos nodos, al igual que los anteriores son clickables.
  - Nodos de tipo **App**, en forma de cajas con la franja superior amarilla. Estos nodos hacen referencia a las aplicaciones por las que pasan los eventos.
  - Nodos de tipo **Logo**, que consisten en una imagen con el logo que representa el origen de los eventos.
- Los **clusters**, que se representan en forma de cajas con la franja superior de color azul-verde, que a su vez contienen cajas de borde discontinuo (nodos internos). La finalidad de estos es poder agrupar los nodos y simplificar el diagrama.
- Los **enlaces** entre los nodos, que representan el flujo que siguen los eventos a lo largo de su ciclo de vida (las distintas etapas por las que se transmite).

4.1.4. Ventanas emergentes

La distinta funcionalidad de la aplicación aparecerá en forma de ventanas emergentes, tal y como se indica en los requisitos. Dentro de las ventanas emergentes podemos distinguir entre dos tipos, las ventanas accesibles desde el diagrama y las accesibles desde el menú desplegable.

Ventanas emergentes accesibles desde el diagrama

Desde los nodos comentados previamente, podemos acceder a distintas ventanas emergentes, estas han sido creadas mediante el *service MatDialog* de Angular que permite crear diálogos (pop-ups) que muestren un componente. Para ello se ha utilizado el método *open()* del *service*.

Siguiendo los requisitos indicados en la Sección 3.1, se han desarrollado tres tipos de ventanas emergentes (pop-ups) accesibles desde los nodos del diagrama.

Para crear las ventanas emergentes descritas en las Figuras 4.3 y 4.4 se ha partido de los *dashboards* generados en *Kibana* que muestran, gráficamente, el número de eventos que pasan por ese nodo y una tabla con la información de estos.

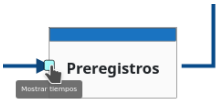
Para mostrar los *dashboards* de *Kibana* en las ventanas emergentes, se ha utilizado el *service* de Angular *DomSanitizer* que permite inyectar código en el lado del cliente de forma segura, es decir, sin código malicioso. De este *service* hemos utilizado el método *bypassSecurityTrustResourceUrl()* que supone que la *URL* es segura, puesto que los *dashboards* de *Kibana* son sólo accesibles si el usuario se ha autentificado en la aplicación de monitoreo o en *Kibana*. Estos *dashboards* se inyectan en el código *HTML* mediante el elemento *iframe*.

Partiendo de estos *dashboards*, sólo ha sido necesario modificar la *URL* de estos para cambiar el rango de tiempo de las gráficas. Como en los requisitos se indicaba que el rango de tiempo sobre el que se generan las gráficas debería poder configurarse desde la ventana emergente, se han incluido dos calendarios para poder establecer la fecha y hora de inicio y de finalización del rango.

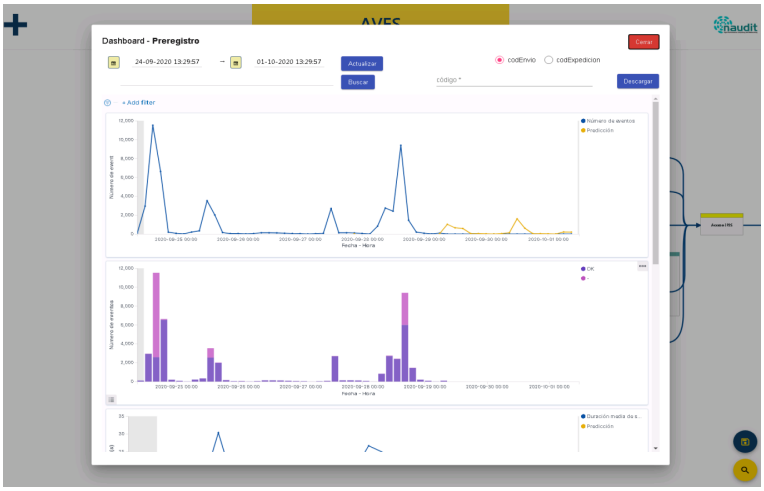
Mediante un botón se podrá actualizar el *dashboard* de *Kibana* para establecer los nuevos tiempos.

La ventana emergente descrita en la Figura 4.5 difiere de las anteriores en que se aprecia un resumen acerca de lo ocurrido con los eventos en la etapa. En este resumen se aprecia una gráfica que muestra, en función del tipo de evento producido (pre-registro, admitido, entregado y devuelto), el número de eventos que ha entrado en la etapa y el número de eventos que ha salido de esta. Para realizar este gráfico se ha utilizado el *framework* de Angular *ngx-charts*.

También se muestra el número de eventos correctos (son aquellos que han entrado y salido de la etapa) y el número de errores producidos (aquellos que han entrado a la etapa, pero que aún no han salido). En el caso de que se hayan producido errores, se mostrará un semáforo en rojo.



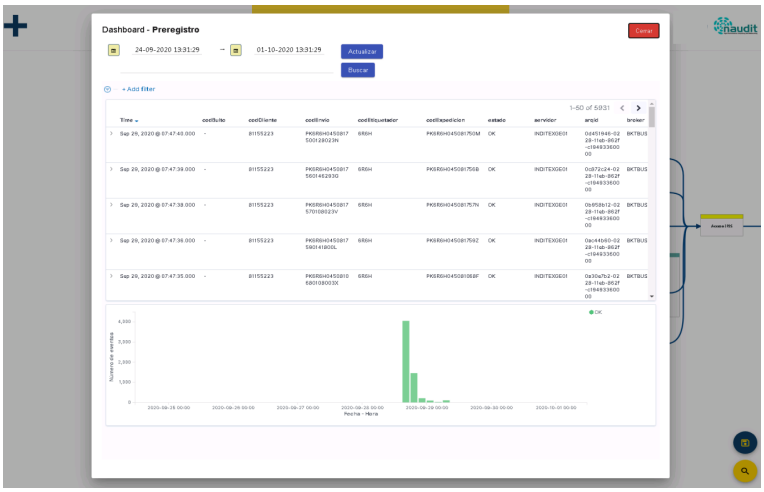
(a) Nodo ejemplo que muestra el dashboard de Kibana



(b) Ejemplo de ventana emergente que aparece al clickar el botón indicado.

Figura 4.3: Nodo y ventana emergente con los *dashboards* de *Kibana* que generan gráficas con el número de eventos que pasa por esa etapa, el número de eventos con salida OK que pasa por esa etapa y una tabla con la información de estos eventos.

(a) Nodo ejemplo que muestra el dashboard de Kibana



(b) Ejemplo de ventana emergente que aparece al clickar el nombre del nodo.

Figura 4.4: Nodo y ventana emergente con los dashboards de Kibana que muestran una tabla con los eventos que pasan por esa etapa y generan gráficas con el número de eventos con salida OK que pasa por esa etapa.

(a) Nodo ejemplo que muestra la gráfica y el dashboard de Kibana

(b) Ejemplo de ventana emergente que aparece al clickar el botón indicado.

Figura 4.5: Nodo y ventana emergente con una gráfica que muestra las entradas y salidas OK de la etapa y un dashboard de Kibana que genera gráficas del número de eventos y de la duración de de cambio de etapa de los mismos, y una tabla con la información de los eventos que han pasado por esta etapa.

Ventanas emergentes accesibles desde el menú de acciones desplegable

Desde la pantalla de inicio, se puede acceder al menú de acciones, tal y como se puede observar en le Figura 4.6(a). Estas acciones son accesibles desde cualquier usuario de la aplicación.

Las distintas acciones accesibles desde el menú, aparecen en ventanas emergentes tras clickarlas.

Alert Manager

La primera opción disponible es la apertura de la *configuración de alarmas o Alert Manager*. Tal y como se indica en el requisito **RF-5** existirán dos versiones, como muestra el menú de la Figura 4.6(b). La Figura 4.7 muestra las dos versiones de la ventana emergente con configuración de alarmas.

La configuración de alarmas de *Alert Manager Administrador*, permite que los usuarios de tipo administrador puedan acceder al gestor de alarmas de administrador y cambiar su configuración. La Figura 4.7(a) muestra la ventana de configuración de alarmas de administrador, las acciones disponibles por estos usuarios son:

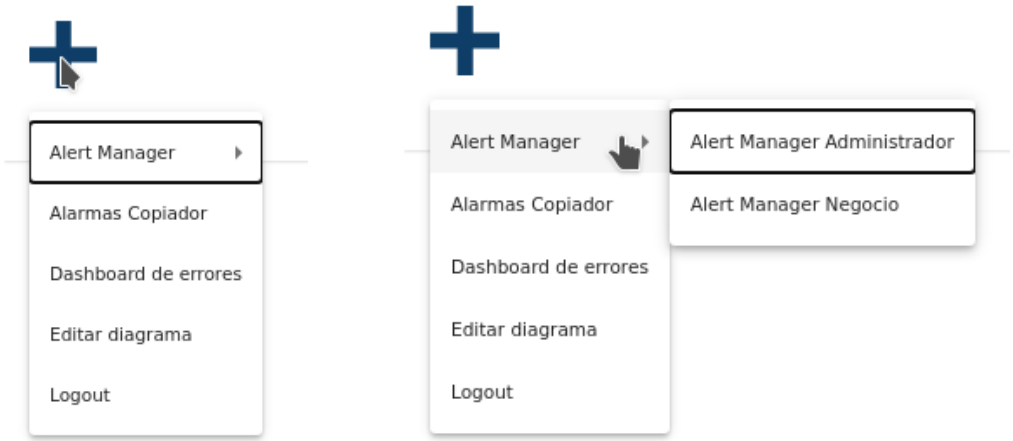
1.– Filtrar las alarmas por nombre o contenido de la configuración.

2.– Desplegar/Replegar la configuración de las alarmas.

3.– Activar/Desactivar alguna o todas las alarmas.

4.– Descartar los cambios realizados en la configuración de alarmas.

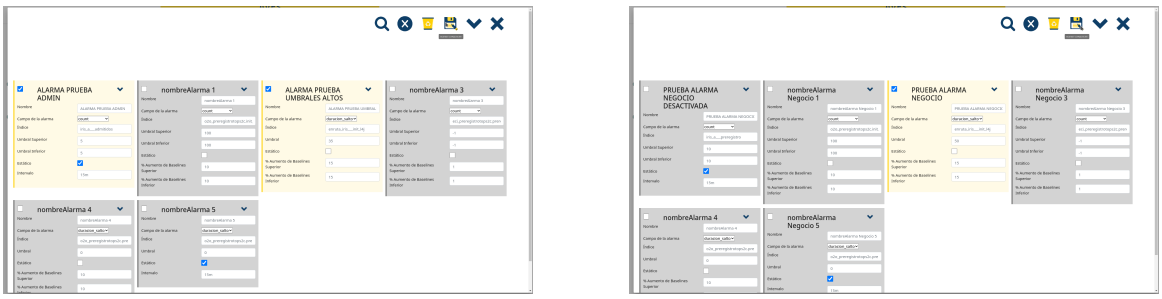
5.– Guardar los cambios realizados en la configuración de alarmas.



(a) Menú de acciones desplegable.

(b) Menú de acciones Alert Manager.

Figura 4.6: Menú desplegable accesible desde la pantalla inicial.



(a) Configuración de alarmas de administrador.

(b) Configuración de alarmas de Negocio.

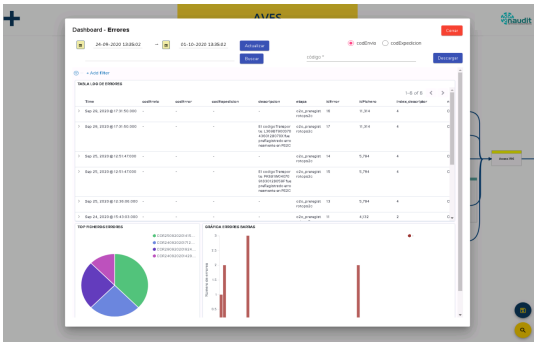
Figura 4.7: Versiones de Administrador y Negocio de la configuración de las alarmas.

6.– Cerrar el gestor de alarmas.

La versión de alarmas de *Alert Manager Negocio*, permite que los usuarios de tipo administrador y consultor puedan acceder al gestor de alarmas de negocio y cambiar su configuración. Mientras que el usuario visualizador podrá realizar las dos primeras acciones explicadas previamente. La Figura 4.7(b) muestra la configuración de alarmas de la versión de Negocio.

Alarmas Copiador

(a) ALARMAS COPIADOR: Dashboard de Kibana con los errores en el copiado de logs.



(b) DASHBOARD DE ERRORES: Dashboard de Kibana con los errores en los eventos.

Figura 4.8: Dashboards de Kibana accesibles desde el menú de acciones.

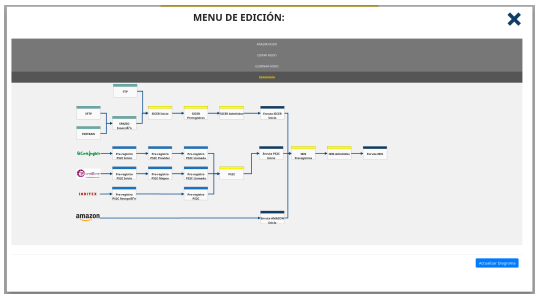
La siguiente opción disponible es abrir el *dashboard* de *Kibana* de los errores de copiador o *Alarmas Copiador*. Esta acción muestra una ventana emergente con el *dashboard* creado en *Kibana* que muestra los errores producidos en el copiador de *logs*. La Figura 4.8(a) muestra la ventana emergente con los errores de copiado de *logs*.

La ventana muestra dos calendarios que permiten modificar el rango de tiempo utilizado para generar el contenido del *dashboard*, mediante el botón de *Actualizar*, se recalcula la *URL* y se actualiza el *dashboard*.

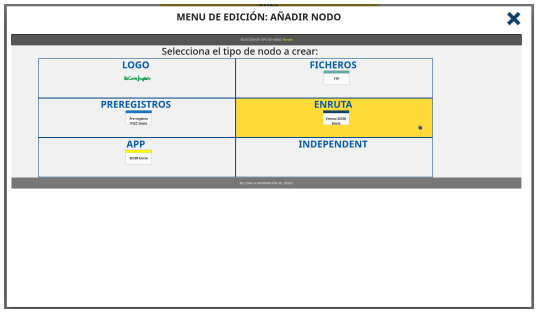
Dashboard de errores

La opción de *Dashboard de errores* muestra una ventana emergente que contiene un *dashboard* de *Kibana* con los errores producidos en la transmisión de los eventos a través de las etapas del diagrama. La Figura 4.8(b) muestra el *dashboard* de *Kibana* con los errores producidos.

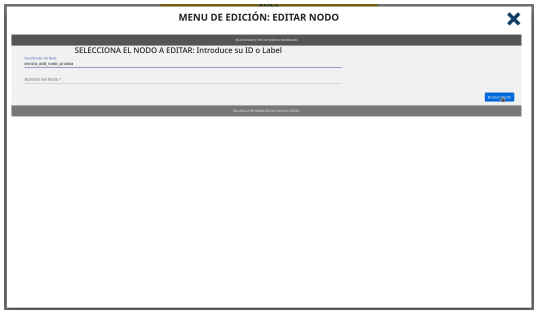
La ventana emergente muestra dos calendarios para modificar el rango de tiempo para generar el contenido del *dashboard*, mediante el botón de *Actualizar*, se recalcula la *URL* y se actualiza el *dashboard*.



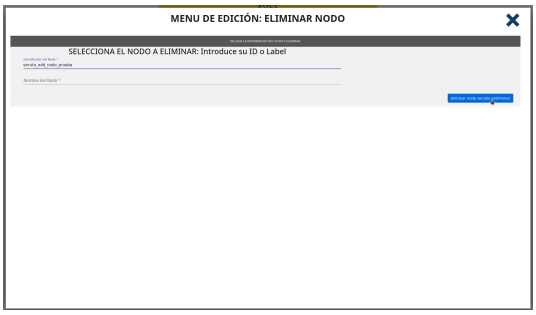
(a) Menú de configuración de nodos.



(b) Menú para añadir un nodo.



(c) Menú de edición de los nodos.



(d) Menú para eliminar un nodo.

Figura 4.9: Ventana emergente con el menú de edición del diagrama de nodos que se observa en la Figura 4.2. Las acciones disponibles son añadir nodo, editar nodo y eliminar nodo.

Editar diagrama

En el caso de acceder a *Editar diagrama*, aparecerá una ventana emergente con el menú de edición del diagrama de nodos. Desde este se podrá editar el diagrama de la pantalla principal, para ello se han desarrollado tres ventanas emergentes. La Figura 4.9 muestra el menú de edición, donde primero aparecen las acciones disponibles y debajo se puede observar y actualizar el diagrama a editar.

La Figura 4.9(b) corresponde a la ventana emergente del menú para *añadir nodo*, primero será necesario seleccionar el tipo de nodo a añadir, tras ello se abrirá una pestaña para la configuración del nuevo nodo, que incluirá un ejemplo de un nodo ya creado a modo de guía. Finalmente, tras rellenar la información correspondiente y guardar, aparecerá una vista preliminar del nodo en el diagrama, desde está se podrán guardar los cambios en el diagrama o cerrar sin modificarlo.

La Figura 4.9(c) muestra el menú de *edición de nodo*, primero se indica el identificador o nombre del nodo a editar, tras ello aparece una pestaña para modificar la configuración del nodo. Finalmente, como en el caso de añadir nodo, aparecerá una versión preliminar del diagrama de nodos donde se podrán guardar los cambios o descartarlos.

La Figura 4.9(d) corresponde al menú para *eliminar un nodo*, primero será necesario indicar el identificador o nombre del nodo a eliminar, tras ello, aparecerá una versión preliminar del diagrama de

nodos, como en los casos anteriores, donde se podrán guardar los cambios o descartarlos.

Logout

La última opción disponible desde el menú de acciones es la opción de *cerrar sesión o Logout*, que desconecta el usuario de su cuenta y redirecciona a la pantalla de login de la aplicación, que se muestra en la Figura 4.1(a).

4.2. Conexión con la base de datos en Elasticsearch

Desde la interfaz web será necesario realizar consultas a la base de datos de *Elasticsearch*. Para realizar estas consultas hemos utilizado el cliente de *Elasticsearch*, mediante la librería *elasticsearch-browser* desarrollada para Angular.

En las Figura 4.3(b), 4.4(b) y 5.8 se puede observar, en la parte superior derecha, un buscador de eventos por código de envío o código de expedición. Este buscador realiza una consulta a *Elasticsearch* sobre el código de envío o expedición introducido, que en caso de existir se podrá descargar un fichero con la información de *logs* sobre el código de envío.

En la Figura 4.5(b) se muestra un gráfico de barras, este se obtiene mediante una consulta a *Elasticsearch*. En la consulta se cuenta el número de eventos que hay guardados en *Elasticsearch* en el nodo de entrada (barra roja) y en la salida (barra azul). El gráfico muestra estos resultados.

También el gestor de alarmas realiza consultas a *Elasticsearch* en varias ocasiones, cuando se introduce o modifica una alarma, la interfaz web, mediante un servidor *PHP* se comunica con *Elasticsearch* e introduce los cambios en las alarmas.

Se realizarán consultas a la base de datos periódicamente - cada 10 minutos - comprobando los índices de las alarmas configuradas desde el gestor, en caso de que el índice contenga entradas en *Elasticsearch*, es decir, en el caso de que se hayan superado los umbrales de las alarmas, se mostrará un icono de alerta en el nodo del diagrama en el que se haya producido la alarma.

En el caso de que el usuario sea administrador, aparecerán en el diagrama iconos de alertas de color amarillo, en los nodos en los que se hayan producido las alarmas configuradas en el gestor de administración y en alertas de color rojo las alarmas del gestor de negocio. La consulta realizada en *Elasticsearch* recupera todas las entradas que contengan en el índice *_alarm_*, obteniendo así las alarmas de negocio y administrador, posteriormente se comprueba el índice de la alarma y en función de este se muestran las alertas de color rojo o amarillo.

Si el usuario que accede a la aplicación es de tipo consultor o visualizador, en el diagrama aparecerán iconos rojos de alertas en los nodos en los que se hayan producido las alarmas configuradas con el

gestor de negocio. Para ello la consulta realizada en *Elasticsearch* busca las entradas que contengan en el índice *_alarm_negocio*, descartando así las alarmas configuradas para el administrador.

4.3. Comunicación con el servidor PHP

Mediante un servidor *PHP*, la aplicación de monitorización realiza las acciones requeridas.

Los ficheros involucrados son:

ldap_connect.php Es uno de los ficheros usados durante el proceso del login a la aplicación, se encarga de conectar con el servidor de autenticación del cliente y comprobar si el usuario y/o la contraseña son correctos, en caso de ser correctos, devuelve los roles asignados al usuario.

auth_ldap.php: Con este archivo se comunica la aplicación para realizar el proceso de autenticación del usuario y contraseña. Para ello se mandan, desde la interfaz, el usuario y la contraseña rellenados en la página del login, posteriormente, este *PHP* se comunica con *ldap_connect.php* y le manda el formulario, luego procesa la respuesta de este y la reenvía a la interfaz web.

Desde la interfaz se accederá a la aplicación y se asignará un rol al usuario, en caso de que la autenticación sea correcta. En caso de que el proceso devuelva un error, se mostrará por pantalla un mensaje informando del mismo.

get_alarmas.php Fichero *PHP* encargado de recuperar las alarmas configuradas en el gestor de alarmas. Estas alarmas se encuentran en los ficheros *JSON conf.json* y *conf_Negocio.json*, que corresponden con las alarmas de administrador y las de negocio respectivamente. Este *PHP* recibe un parámetro en forma de *string*, indicándole el tipo de alarmas a devolver, entre negocio y administrador.

Desde la interfaz web, cuando se abren cualquiera de los gestores de alarmas, se realiza una consulta a este *PHP* para recuperar las alarmas. Según se acceda al gestor de administrador o negocio, la interfaz manda al *PHP* un *string* indicándole el tipo de alarmas a mostrar (entre negocio y administración), y según este, el *PHP* devuelve la lista de alarmas de administrador o de negocio.

save_alarmas.php Este fichero se encarga de guardar las alarmas configuradas. Este *PHP* recibe dos parámetros, uno indicándole el tipo de alarmas a guardar, entre negocio o administrador, y un *array* con las alarmas a guardar en el fichero *JSON* correspondiente, *conf.json* o *conf_Negocio.json*.

También, desde este fichero *PHP*, se ejecuta un script configurado para crear o modificar en *Elasticsearch* las alarmas indicadas en los ficheros *JSON*.

Desde la interfaz web, según el gestor que se haya abierto, se mandará un *string*, indicando

el tipo de alarma a guardar, y un *array* con la última configuración guardada de las alarmas.

get_flow.php Fichero encargado de recuperar los nodos del diagrama. Se encarga de devolver el fichero *JSON flujos.json* que contiene la información correspondiente a los nodos del diagrama.

La interfaz se comunica con este *PHP* al cargar el diagrama en la pantalla inicial, ya que recibe los nodos (junto con su información) del diagrama. También se realiza la comunicación al abrir el editor del diagrama.

save_flow.php Este fichero recibe como parámetro una lista con los nodos del diagrama y la información de estos. Primero comprueba que el formato de los nodos es el correcto y en ese caso, actualiza el fichero *JSON* con los nodos, *flujos.json*.

La interfaz realiza una llamada a este fichero desde el editor del diagrama, tras guardar los cambios y le envía la nueva lista de nodos.

5

Pruebas y resultados

El proyecto se desarrollará de forma incremental. Para ello se han desarrollado distintas versiones de la aplicación con la funcionalidad requerida, a modo de prototipo, y se han mostrado al cliente para obtener los cambios que considere en la interfaz o funcionalidad de la aplicación, o para recibir su confirmación.

5.1. Pruebas realizadas

En este proyecto se han realizado pruebas de caja negra, que verifican la funcionalidad del software, sin tener en cuenta la implementación del código.

Para realizarlas, primero habrá que acceder a un sistema de la red interna del cliente y abrir el navegador en la dirección *URL* de la aplicación. Al hacerlo aparecerá la ventana con el login de la aplicación y para acceder a la funcionalidad será necesario autenticarse.

Tras realizar la autenticación, accederemos a la pantalla principal, desde la cual podremos acceder a la funcionalidad de la aplicación.

Diagrama

Desde el diagrama, tal y como hemos explicado en el Capítulo 4, podremos acceder, al clickar en los nodos o enlaces, a las ventanas emergentes que nos muestran la información de los nodos o de los flujos respectivamente.

Desde estas ventanas, comprobaremos que si muestran los *dashboards* de *Kibana* adecuados y si se modifican al cambiar el rango de tiempo.

También desde estas ventanas comprobaremos si la búsqueda y descarga de *logs* de eventos por código de envío funciona, en el caso de que el código sea correcto. O bien muestra mensaje de error en el caso de ser un código erróneo.

Alert Manager

Accederemos al Alert Manager desde el menú desplegable mostrado en la Figura 4.6.

Con un usuario de tipo administrador, accederemos al Alert Manager Administrador para editar las alarmas existentes y comprobar si el guardado de estas funciona correctamente.

Posteriormente, accederemos al Alert Manager Negocio para modificar las alarmas de Negocio, y comprobaremos si se guardan correctamente.

Finalmente, desde el diagrama podremos comprobar si las alarmas se han configurado correctamente en *Elasticsearch*, ya que se podrán observar iconos de alertas en los nodos en los que se produzcan.

Alarmas Copiador y Dashboard de errores

Desde el menú desplegable, podremos acceder a las ventanas emergentes que se muestran en la Figura 5.8 y desde estas comprobaremos si los *dashboards* de *Kibana* se actualizan al cambiar el rango de tiempo.

Editar diagrama

Accederemos a la ventana de edición del diagrama desde el menú desplegable. Desde esta ventana procederemos a crear un nuevo nodo y comprobaremos si el proceso se ha realizado correctamente desde el diagrama de edición.

Posteriormente, editaremos el nodo previamente creado y comprobaremos si se ha guardado correctamente desde el diagrama.

A continuación, eliminaremos el nodo creado y editado. Comprobaremos si la acción funciona correctamente desde el diagrama de edición.

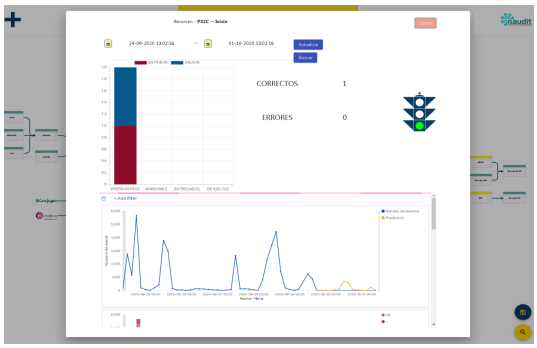
5.2. Resultados obtenidos

Para comenzar las pruebas, se introducen los credenciales en el login que muestra la Figura 4.1 y, en caso de ser válidos, accederemos a la aplicación. En caso de ser erróneos, como es el caso de la Figura 4.1(b), se mostrará un mensaje de error y se mantendrá la ventana del login hasta autenticarse correctamente.

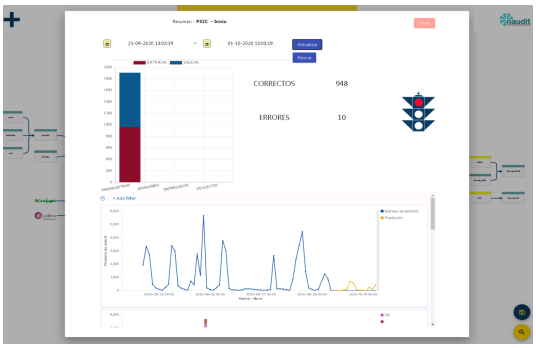
Tras acceder a la pantalla principal, que se puede observar en la Figura 4.2, podremos acceder a la funcionalidad de la aplicación desde el diagrama o desde el menú desplegable, tal y como comentamos en el Capítulo 4.

Diagrama

Para comenzar las pruebas, accederemos a las ventanas emergentes del diagrama. Existen tres tipos, y en todos ellos, para comprobar que se crean las gráficas y que se carguen los *dashboards* de *Kibana* correspondientes, procederemos a cambiar el rango de tiempo a analizar.



(a) Ventana emergente con el resumen de la última semana.



(b) Ventana emergente con el resumen de los últimos 10 días.

Figura 5.1: Ventanas emergentes con el resumen de los acontecido en las etapas por las que pasa el evento. Se muestra una gráfica con el número de eventos que entran frente a los que salen y se observa un *dashboard* de *Kibana* con la información de los eventos.

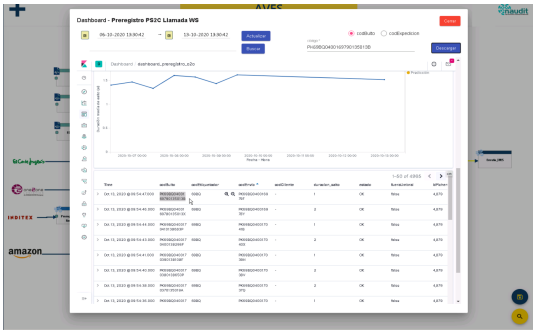
Comenzamos comprobando la correcta creación de las gráficas en la ventana emergente que muestra un resumen de lo ocurrido en las etapas, para ello, tenemos que clickar el enlace del nodo *Enruta PS2C*, tal y como se muestra en la Figura 4.5(a) se abrirá una ventana emergente con la información de la última semana, la Figura 5.1(a) muestra el resumen de la última semana. Si procedemos a cambiar el rango de tiempo a 10 días, podemos observar como cambia el resumen mostrado en esta ventana, la Figura 5.1(b) muestra esta variación en el resumen.

(a) Ventana emergente con la información de la última semana.

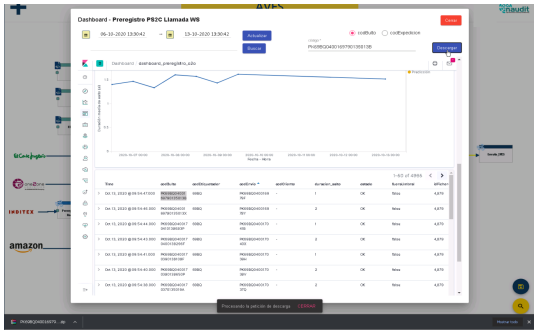
(b) Ventana emergente con la información de los últimos 10 días.

Figura 5.2: Ventanas emergentes con los *dashboards* de *Kibana* con el número de eventos que pasa por esa etapa, el número de eventos con salida OK que pasa por esa etapa y una tabla con la información de estos eventos.

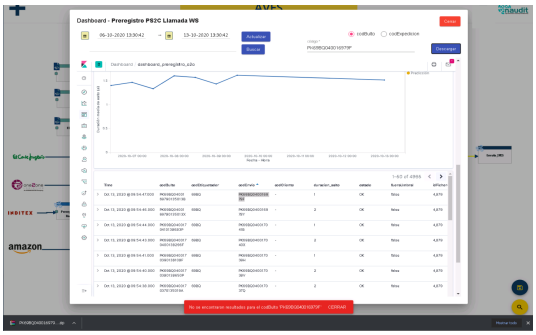
Además desde esta ventana, podremos realizar la búsqueda por código de envío, de los eventos



(a) Búsqueda por código de envío.



(b) Resultado de la búsqueda por código de envío.



(c) Mensaje de error al buscar un código que no existe.

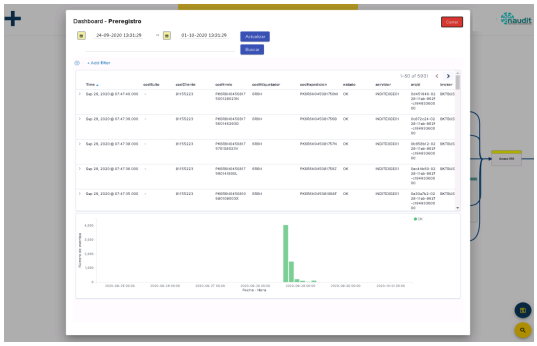
Figura 5.3: Figuras que muestran el proceso de búsqueda por código de envío.

transmitidos en las etapas. Primero descargaremos un código de envío existente, elegiremos uno de los que se encuentren en tabla del *dashboard* de *Kibana* y descargamos los *logs* asociados a este código. Las Figuras 5.3(a) y 5.3(b) muestran este proceso.

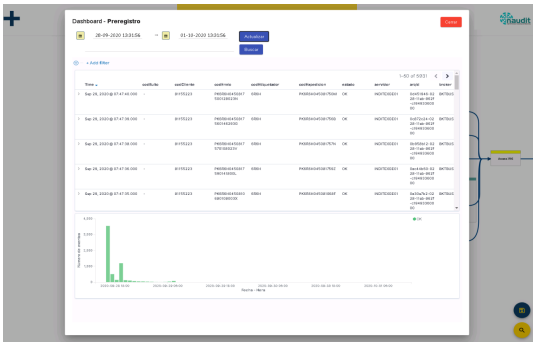
También probaremos la búsqueda para un código de envío inexistente, la Figura 5.3(c) muestra el mensaje de error al intentar descargar los *logs* de un código que no existe.

A continuación, clickaremos uno de los enlaces de los nodos, en este caso el enlace al nodo *Pre-registro*. Se abrirá una ventana emergente con los *dashboards* de *Kibana* que muestran dos gráficas de el número de eventos y una tabla con su información. La Figura 5.2(a) muestra está ventana emergente, con la información de la última semana. Si procedemos a cambiar el rango de tiempo estudiado a 10 días, podemos observar que se ha producido una variación en el *dashboard* de *Kibana*, tal y como muestra la Figura 5.2(b)

Finalmente clickaremos el nodo *Pre-registro*. Se abrirá una ventana emergente con un *dashboard* de *Kibana* en que el que se observa una tabla con la información de los eventos de esa etapa y una gráfica con el número de eventos de la última semana. Si cambiamos el rango de tiempo a analizar a 10 días, observamos como se produce un cambio en el *dashboard* de *Kibana*. La Figura 5.4(a) muestra la información de la última semana, mientras que la Figura 5.4(b) muestra la información de los últimos 10 días.



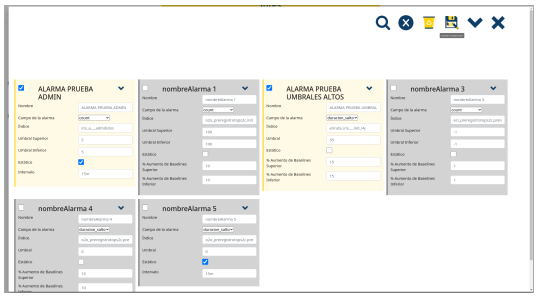
(a) Ventana emergente con la información de la última semana.



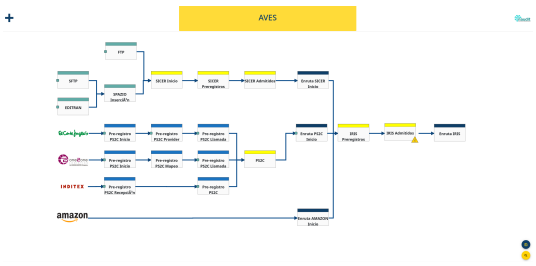
(b) Ventana emergente con la información de los últimos 10 días.

Figura 5.4: Ventanas emergentes con los *dashboards* de *Kibana* que muestran una tabla con los eventos que pasan por esa etapa y generan gráficas con el número de eventos con salida OK que pasa por esa etapa.

Alert Manager



(a) Ventana emergente con la configuración de las alarmas administrador.



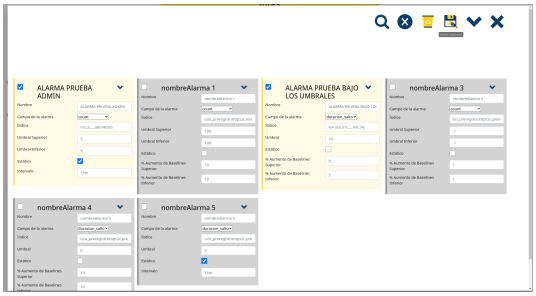
(b) Diagrama tras configurar las alarmas de administrador.

Figura 5.5: La primera Figura corresponde con la ventana emergente para gestionar las alarmas de tipo administrador, mientras que la segunda Figura corresponde al diagrama de nodos tras configurar las alarmas de administrador.

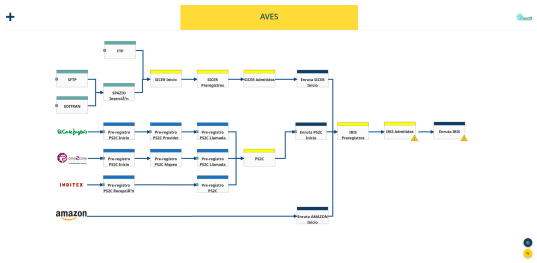
Accederemos con un usuario de tipo administrador a la aplicación y, mediante el menú desplegable del lateral superior izquierdo de la pantalla principal, clickaremos en la opción de Alert Manager y seleccionaremos la opción Alert Manager Administrador. La Figura 5.5(a) muestra la configuración de alarmas de administrador, al guardar esta configuración, en el diagrama aparecerán dos tipos de iconos de alertas: alertas en amarillo (alarmas configuradas en Alert Manager Administrador) y alertas en rojo (alarmas configuradas en Alert Manager Negocio). La Figura 5.5(b) muestra el diagrama de nodos con la configuración que muestra la Figura 5.5(a).

De modo que accedemos al Alert Manager Administrador y modificamos el nombre y el umbral de una de las alarmas. La Figura 5.6(a) muestra la configuración de las alarmas tras realizar un cambio en la alarma de nombre *Alarma Umbrales Bajos*, en la cual cambiamos el umbral de 130 a 1 (con el fin de que la alarma salte con más facilidad). Tras guardar la nueva configuración, cerramos el gestor de

alarmas y, tal y como muestra la Figura 5.6(b), podremos observar como se ha producido un cambio en el diagrama apareciendo una nueva alerta, de color amarillo, en el nodo *Enruta IRIS*.



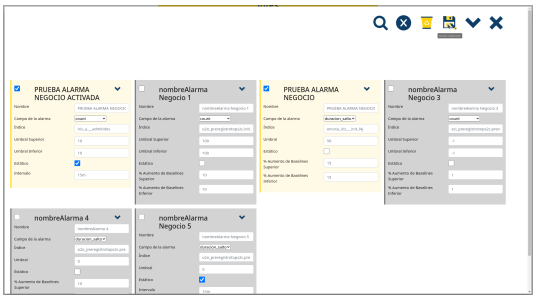
(a) Ventana emergente con la nueva configuración de las alarmas administrador.



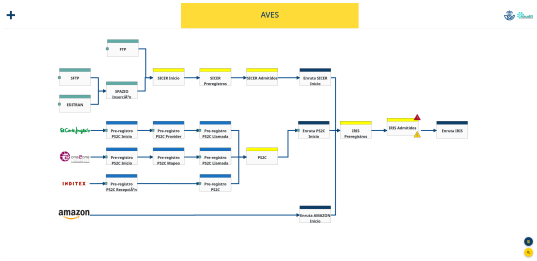
(b) Diagrama tras la nueva configuración las alarmas de administrador.

Figura 5.6: La primera Figura corresponde con la ventana emergente para gestionar las alarmas de tipo administrador tras realizar un cambio en ellas, mientras que la segunda Figura corresponde al diagrama de nodos tras configurar las alarmas de administrador.

Tras estos cambios en las alarmas administrador, accederemos al gestor de alarmas de negocio, cuya configuración se puede observar en la Figura 4.7(b) y cambiaremos la configuración de la alarma activa para que el umbral sea más alto y la alarma no se produzca con tanta facilidad. La Figura 5.7(a) corresponde a la nueva configuración de las alarmas de negocio. Tras guardar los cambios en las alarmas, podemos comprobar como se produce un cambio en el diagrama, que pasa de tener una alerta de tipo negocio (el icono rojo), tal y como se muestra en la Figura 5.6(b) a no presentar ninguna alerta, como se puede observar en la Figura 5.7(b).



(a) Ventana emergente con la nueva configuración de las alarmas de negocio.



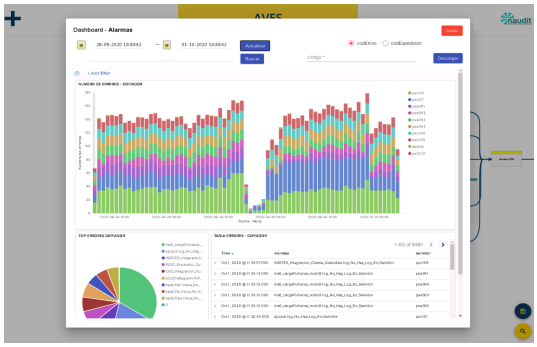
(b) Diagrama tras la nueva configuración las alarmas de negocio.

Figura 5.7: La primera Figura corresponde con la ventana emergente para gestionar las alarmas de tipo negocio tras realizar un cambio en ellas, mientras que la segunda Figura corresponde al diagrama de nodos tras guardar la configuración de las alarmas de negocio.

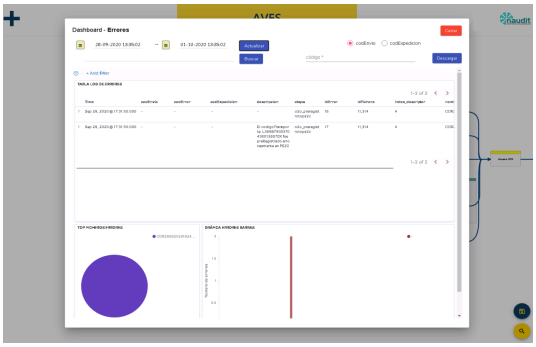
De modo que queda comprobado que el gestor de alarmas y el diagrama siguen su comportamiento esperado.

Alarmas Copiador y Dashboard de errores

Continuamos desde el menú desplegable con la siguiente opción disponible, acceder a la ventana emergente con el *dashboard* de *Kibana* de Alarmas copiador, la Figura 4.8(a) muestra el *dashboard* generado con un rango de tiempo de una semana. Si cambiamos el rango de tiempo y lo reducimos a tres días, vemos como las alarmas de copiado se reducen en gran medida, tal y como se puede observar en la Figura 5.8(a).



(a) ALARMAS COPIADOR: Dashboard de Kibana con los errores en el copiado de logs de los últimos 3 días.



(b) DASHBOARD DE ERRORES: Dashboard de Kibana con los errores en los eventos de los últimos 3 días.

Figura 5.8: Dashboards de Kibana accesibles desde el menú de acciones con un rango de tiempo de 3 días.

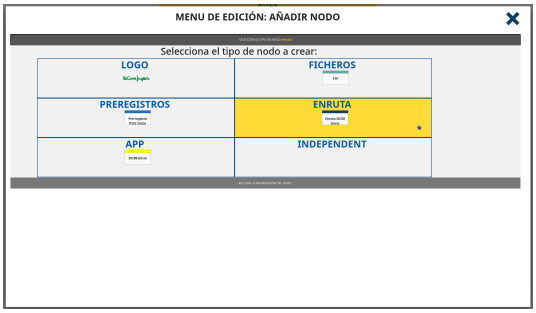
Si realizamos la misma acción en la ventana emergente (accesible desde el menú desplegable) del Dashboard de errores, podemos observar que también se reducen los errores producidos en la transmisión de los eventos. La Figura 4.8(b) muestra el *dashboard* de *Kibana* con los errores producidos en la última semana, mientras que la Figura 5.8(b) muestra los errores producidos los últimos 3 días.

Por lo que podemos decir que la funcionalidad de las ventanas emergentes de Alarmas Copiador y de Dashboard de errores tiene el comportamiento esperado.

Editar diagrama

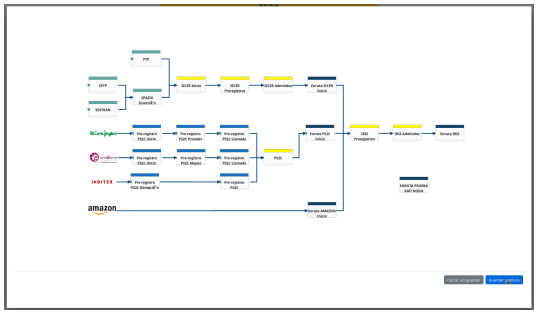
Por último, accederemos al menú de edición del diagrama mediante el menú desplegable. Desde esté, procederemos a añadir un nuevo nodo, en este caso seleccionaremos el tipo Enruta. A continuación rellenaremos la información del nodo, la Figura 5.9(b) muestra la configuración que tendrá el nodo. Finalmente guardaremos el nodo y accederemos a la vista preliminar del diagrama, para comprobar el resultado de los cambios sobre el diagrama general y guardaremos estos cambios. La Figura 5.9 muestra todo el proceso de añadir un nuevo nodo y la Figura 5.9(d) muestra el menú de edición del diagrama, tras añadir un nuevo nodo.

Tras añadir y comprobar que el nodo se ha creado correctamente, procederemos a editar el nuevo nodo creado. Para ello primero introducimos el identificador del nodo y lo buscamos, en nuestro caso

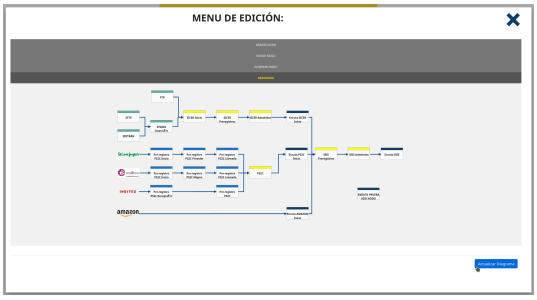


(a) AÑADIR NODO: Selección de tipo de nodo.

(b) AÑADIR NODO: Rellenar la información del nodo.



(c) AÑADIR NODO: Vista preliminar del diagrama.



(d) AÑADIR NODO: Guardado y diagrama final actualizado.

Figura 5.9: EDICIÓN DEL DIAGRAMA: AÑADIR NODO.

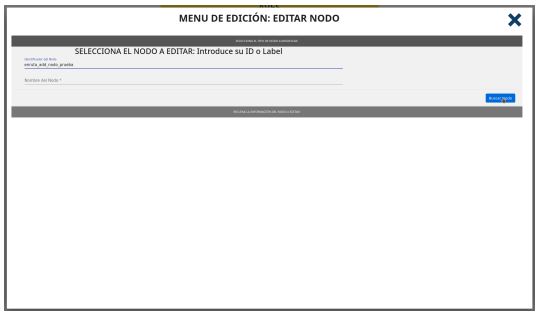
como el nodo existe, podremos editar la información del nodo. Cambiaremos el nombre del nodo y el nodo antecesor a este, por lo tanto si el cambio se realiza correctamente, el nodo aparecerá en otra posición y con un nuevo nombre.

La Figura 5.10 muestra como se ha realizado todo el proceso de edición del nuevo nodo creado. Como se aprecia en las Figuras 5.10(c) y 5.10(d), la edición del diagrama se ha realizado de forma correcta.

Hasta ahora hemos comprobado que la funcionalidad de añadir y editar nodo funciona correctamente, por último, procederemos a realizar la eliminación del nodo creado. Para ello accederemos al menú de eliminar nodo e introduciremos el identificador del nodo creado. A continuación buscaremos el nodo y tras encontrarlo, podremos acceder a la vista preliminar del diagrama para comprobar que el nodo introducido se ha eliminado correctamente y guardar los cambios.

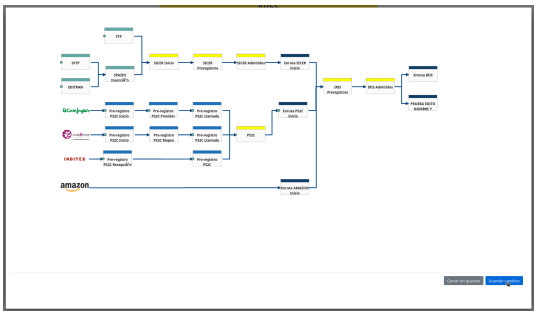
La Figura 5.11 muestra todo el proceso seguido para la eliminación del nodo creado. Podemos comprobar que la eliminación del nodo ha funcionado correctamente observando las Figuras 5.11(c) y 5.11(d) que muestran el resultado del diagrama principal tras realizar la eliminación del nodo creado.

Por lo tanto, tras realizar estas acciones, hemos comprobado que la funcionalidad de la edición del diagrama sigue el comportamiento esperado.

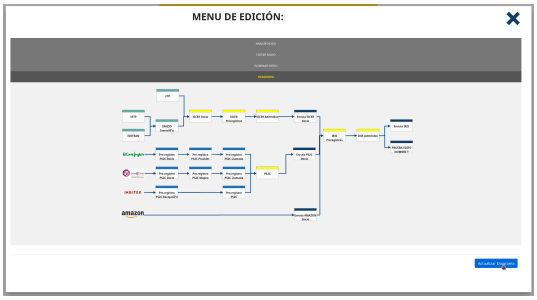


(a) EDITAR NODO: Búsqueda, por nombre o identificador, del nodo a editar.

(b) EDITAR NODO: Rellenar la nueva información del nodo.

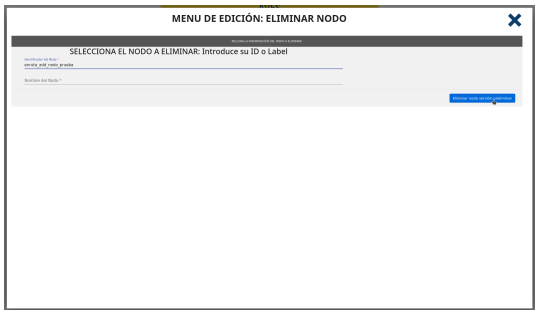


(c) EDITAR NODO: Vista preliminar del diagrama.

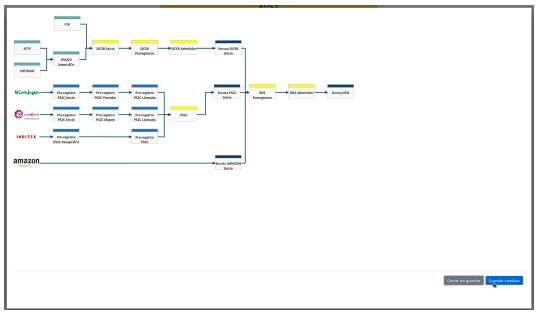


(d) EDITAR NODO: Guardado y diagrama final actualizado.

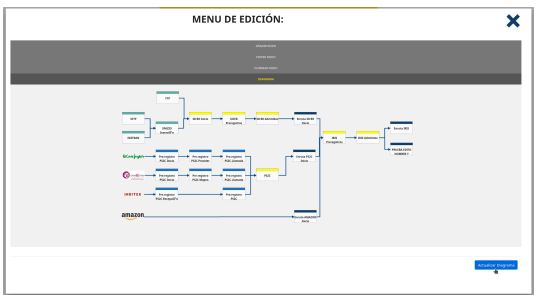
Figura 5.10: EDICIÓN DEL DIAGRAMA: EDITAR NODO.



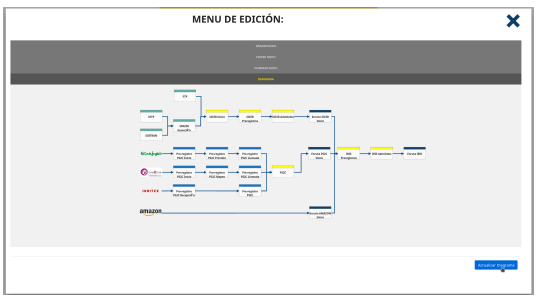
(a) ELIMINAR NODO: Búsqueda, por nombre o identificador, del nodo a editar.



(b) ELIMINAR NODO: Rellenar la nueva información del nodo.



(c) ELIMINAR NODO: Vista preliminar del diagrama.



(d) ELIMINAR NODO: Guardado y diagrama final actualizado.

Figura 5.11: EDICIÓN DEL DIAGRAMA: ELIMINAR NODO.

6

Conclusiones

6.1. Conclusiones del trabajo realizado

Este trabajo me ha permitido desarrollar distintas partes de un proyecto, desde el análisis y establecimiento de los requisitos hasta el diseño y desarrollo de la aplicación.

La aplicación cumple con todos los requisitos del cliente, para ello ha sido necesario familiarizarse con la herramienta Angular y, en especial, con la librerías mencionadas en la Sección 2.1 para poder personalizar el diagrama de nodos. Se ha utilizado esta herramienta puesto que el cliente estaba interesado en que la aplicación mostrara una ventana inicial, sobre la cual fueran apareciendo ventanas emergentes.

El diseño de esta aplicación se ha centrado en las necesidades del cliente y dado que su interés primordial es conocer el estado de los eventos transmitidos y analizar el flujo de estos, se ha desarrollado una aplicación con un diseño simple e intuitivo, que mediante la aparición de alarmas en el diagrama o los *dashboards* de *Kibana*, facilita el trabajo de los analistas de nuestro cliente (la compañía logística española que nos contrató).

El diseño modular de la aplicación permite que se pueda modificar o añadir funcionalidad a una componente, sin afectar al comportamiento del resto. Esto facilita el mantenimiento futuro además de reducir la complejidad del código.

Gracias a este trabajo he aprendido la importancia de una buena comunicación con el cliente, para garantizar su satisfacción con el producto solicitado. Además, he aprendido la necesidad de una buena coordinación con los compañeros que desarrollan el proyecto. Todo ello me ha permitido aplicar los conocimientos estudiados en la carrera para solucionar un problema real.

6.2. Trabajo futuro

Actualmente esta aplicación está en uso en un conjunto de sistemas de una empresa logística española, sin embargo, se podría extender su uso al resto de sus sistemas para monitorizar una mayor

cantidad de eventos. Si todos los eventos son iguales, únicamente sería necesario crear los ficheros *JSON* correspondientes a los diagramas y añadir una nueva componente, a la aplicación, que permita navegar entre los distintos diagramas de flujos disponibles.

También existe la posibilidad de gestionar el flujo de eventos de otro tipo, para ello sería necesario definir una nueva estructura de los nodos.

Otra de las mejoras que se podrían realizar a la aplicación sería incorporar un servicio de mensajería, encargado de notificar a los analistas las alertas producidas y una tabla que incluya un listado con las alertas producidas en las últimas horas.

Bibliografía

[1] Seshadri, S.: *Angular: Up and Running: Learning Angular, Step by Step*. O'Reilly (2018).

[2] Cherny, B.: *Programming TypeScript: Making Your JavaScript Applications Scale*. O'Reilly (2019).

[3] Correa, M., García, J. R. y Tabanera A.: Comercio electrónico y hábitos de consumo en España: la importancia de la banca on-line. *Observatorio Economía Digital* BBVA RESEARCH, España. **1** (https://www.bbvaresearch.com/) (2015).

[4] PostgreSQL, Documentation, Welcome to the PostgreSQL Wiki!. (https://www.postgresql.org/).

[5] Elasticsearch, Elastic Stack and Product Documentation, Elasticsearch: Store, Search, and Analyze. (https://www.elastic.co/guide/en/elastic-stack/current/index.html).

[6] Elasticsearch, Elastic Stack and Product Documentation, Kibana: Explore, Visualize, and Share. (https://www.elastic.co/guide/en/kibana/current/index.html).


UAM
UNIVERSIDAD AUTONOMA
DE MADRID